




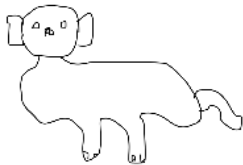
# Homework 3

Student : Miro Filomeno Davide

## 1) Dataset

---

This homework try to solve the problem of **domain adaptation** using a neural network that implement the model of **DANN** using the alexnet model available pretrained in pytorch libray.The dataset used for the analysis is the **PACS** dataset that has become a benchmark for the model related to domain adaptation.This dataset is divided in 4 different domains: **Photo** ,**Art painting**,**Cartoons**,**Sketch**. Each of them is splitted in 7 different classes: **dog,elephant,giraffe,guitar,horse,house** and **person**. Below there are an example of representation of same class **dog** in different domanins.

Photo	Art paint	Cartoons	Sketch
			

## 2) Implementing the model

---

The model is buildt using pretrained components of **Alexnet** neural network.The classifier and discriminator has the same pretrained weight of alexnet classifier and the feature extractor's layers are copied from convolutional layers of alexnet.The new elements is the reveral layer that in backward phase of discriminator's learning change the sign of gradient.This trick implement a min/max game that try to learn feature extractor to provide a feature mapping in a space in which domains distributions overlaps.Given the forward code:

```

def forward(self, input_data, alpha):
    feature = self.features(input_data)
    feature = self.avgpool(feature)
    feature = torch.flatten(feature, 1)
    if alpha is not None:
        # gradient reversal layer (backward gradients will be reversed)
        reverse_feature = ReverseLayerF.apply(feature, alpha)
        discriminator_output = self.domain_classifier(reverse_feature)
        return discriminator_output
    else:
        # do something else
        class_outputs = self.class_classifier(feature)
        return class_outputs

```

If you specify **alpha** you learn the discriminator otherwise you learn the classifier.

### 3) Implementing training with DANN

---

The training phase of **DANN** is implemented using **photo** domain as **source domain** and **art painting** as **target domain**. With source dataset you learn classifier and discriminator, with target you learn the discriminator and test the performance of model. As optimization strategy I used **SGD with momentum = 0.9** and **weight decay = 5e-5**. As loss functions I used **cross entropy loss** both for discriminator and classifier. The code of training phase is the following:

```

net = DANN()
net.class_classifier[6] = nn.Linear(4096, 7) # nn.Linear in pytorch is a

optimizer = optim.SGD(net.parameters(), lr=LR, momentum=MOMENTUM, weight_decay=WEIGHT_DECAY)

for p in net.parameters():
    p.requires_grad = True

loss_class = nn.CrossEntropyLoss()
loss_domain = nn.CrossEntropyLoss()

net = net.cuda()
loss_class = loss_class.cuda()
loss_domain = loss_domain.cuda()

net = net.to(DEVICE) # this will bring the network to GPU if DEVICE is cuda
cudnn.benchmark # Calling this optimizes runtime

for epoch in range(NUM_EPOCHS):

```

```

iter_target = iter(target_dataloader)
for xs,ys in source_dataloader:
    net.zero_grad()
    try:
        xt,yt = next(iter_target)
    except:
        iter_target = iter(target_dataloader)
        xt,yt = next(iter_target)

    xs = xs.to(DEVICE)
    ys = ys.to(DEVICE)
    xt = xt.to(DEVICE)
    ysd = torch.zeros(BATCH_SIZE).long().to(DEVICE)
    yt = torch.ones(BATCH_SIZE).long().to(DEVICE)

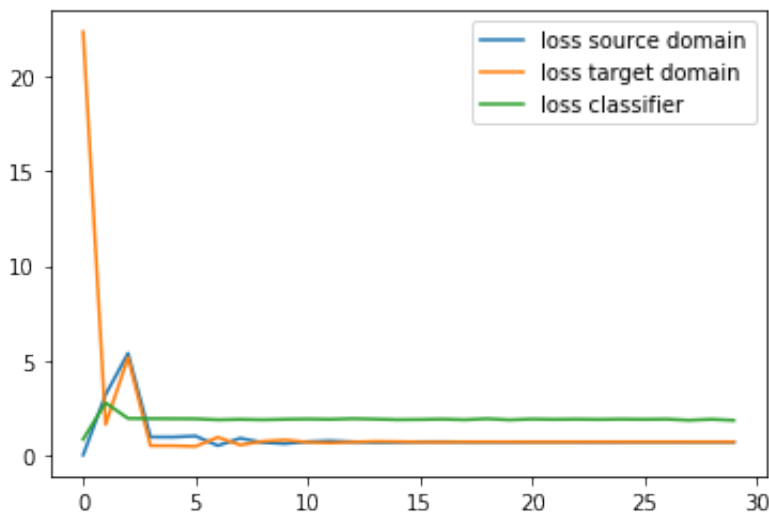
    class_output = net(input_data=xs,alpha = None)
    err_s_label = loss_class(class_output, ys)
    err_s_label.backward()

    domain_output = net(input_data=xs,alpha=ALPHA)
    err_s_domain = loss_domain(domain_output, ysd)
    err_s_domain.backward()

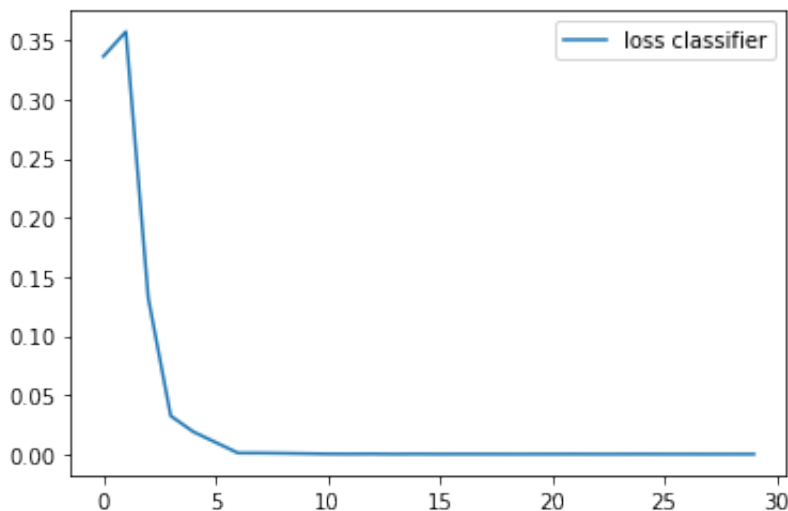
    domain_output = net(input_data=xt,alpha=ALPHA)
    err_t_domain = loss_domain(domain_output,yt)
    err_t_domain.backward()
    optimizer.step()

```

As you can see the training phase is splitted in 3 sub-phases: train the classifier and compute the gradients with `backward()`, train discriminator with both source and target data and at the end update weights. To learn the model with **no domain adaptation** I kept only the first phase (classification on source domain) because the forwarding if you not specify alpha will learn only classifier and feature extractor and not pass through the discriminator. I first train the model with domain adaptation using as hyperparameters the same that I optimized in the homework 2 (**LR = 1e-2, batchsize = 256, numepochs = 30, step\_size = 20, gamma = 0.1**) with the addition of **alpha = 0.1**. The resulting model provide an accuracy on target dataset equal to **0.22**. This is the graph of the loss:



I also try to learn the model without domain adaptation, using the learning strategy that I described, using the same hyperparameters used with DANN (without alpha). The model provides an accuracy on target dataset equal to **0.44**

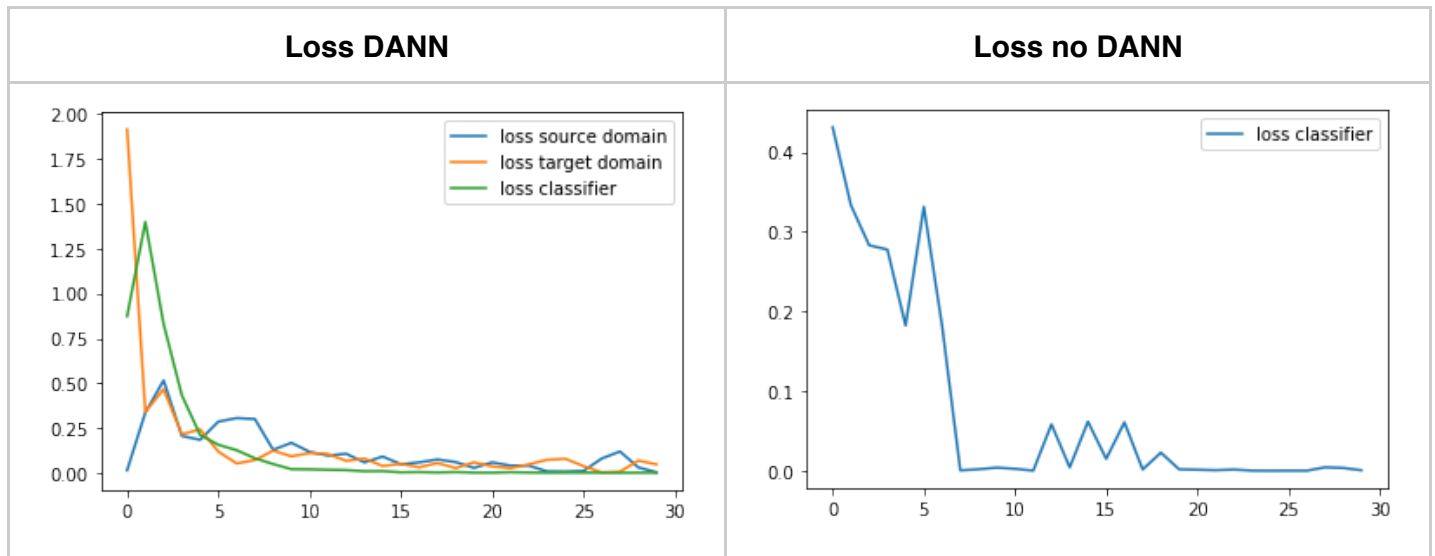


It seems that Alexnet with no DANN performs better but you should take in consideration that there is no hyperparameter tuning of DANN so it's important to select best set of hyperparameters and an important tool to do this is grid search

## 4) Cross domain validation

To obtain better performance with DANN I used grid search for validating hyperparameters on both cartoon and art painting. The hyperparameter that I decided to tune are **learning rate = [1e-3, 1e-2, 1e-1]**, **batch size = [64, 128, 256]** and for DANN also **alpha = [1e-2, 1e-3, 1e-1]**. The number of epochs at each training iteration is set to 5 to colab time limitations. To validate the hyperparameters both with cartoon set and sketch set at each training iteration I train 2 models one using cartoon set as target and the other one using sketch. The source is photo in both cases. The result of grid search gives for DANN as best hyperparameters set of **learning rate = 0.01**, **alpha = 0.01** and **batch size = 256** and for no DAN **learning rate = 0.01** and

**batch size = 64**. The resulting models are trained using as source domain photo and art painting as target domain and provides an accuracy of **0.50** for DANN and **0.48** for model with no domain adaptation. The graphs of losses trends are shown below:



The code relative to this homework respects the template provided in homework 2, but there are 5 new cells:

- **grid search** to run cross domain validation for DANN.
- **grid search no DANN** to run cross domain validation for no DANN.
- **train** to run training of DANN. If you do not run grid search cell you can set hyperparameters from the first cell.
- **train no DANN** to run training of no DANN. If you do not run grid search cell you can set hyperparameters from the first cell.